Contents Lecture 8

- Review of the heap data structure
- Overview of array-based heap
- Overview of Fibonacci heap
- Hollow heap
Review of the heap data structure

- 
- \((key, value)\) pairs are stored
- Primarily used for priority queues
- Operations:
  - make heap
  - insert pair
  - change key
  - min
  - delete min
- Efficient search is not supported
Overview of array-based heap data structure

- To store up to \( s \) pairs \((\text{key}, \text{value})\), an array indexed from 1 to \( s \) is used.
- The root is stored at index 1.
- Let \( k_j \) denote key of pair stored at index \( j \).
- The heap order means that \( k_j \leq k_{2j} \) and \( k_j \leq k_{2j+1} \).
- But nothing about \( k_{2j} \) vs. \( k_{2j+1} \).
Operations on array based heap

- Assume the heap contains $n$ pairs
- To delete the min pair it is saved and the pair at index $n$ is moved to index 1
- This pair is then moved down which takes $O(\log n)$ time
- A new pair is inserted at index $n + 1$
- The new pair is then moved up which also takes $O(\log n)$ time
- Changing the priority takes $O(\log n)$ time as well
Fibonacci heaps

- Uses trees instead of an array
- Worst-case constant time to insert a new \((key, value)\) pair
- Basic idea of insert: create a new tree and check if it is the minimum
- Amortized constant time to decrease a key
- Basic idea of decrease-key: remove it from the parent and make it a new root and possibly make additional updates
- Amortized \(O(\log n)\) time to remove minimum
- Each tree node uses five pointers, an integer and a boolean
Hollow heaps

- Simpler and better than Fibonacci heaps
- A disadvantage is that some nodes have no data and still consume memory
- They can be cleaned away when needed though
- This is research published in 2015 and 2017 by Dueholm Hansen, Tarjan, Kaplan and Zwick
- Hollow heaps also uses trees, just as Fibonacci heaps
Nodes and elements

- A node is a tree node in the hollow heap
- An element is the data stored in the heap: a \((key, value)\) pair
- A node with an element is full
- A node with no element is hollow
Hollow nodes

- An element can be removed from a node which then becomes a hollow node
- A node is not the element but instead has a pointer to an element (or null)
- Thus a node cannot be an element — only point to an element
- A node also has a key: identical to the element’s or to the key of the element the node previously had
- A hollow node never gets a new element
- Hollow nodes which are children of the minimum node are thrown away when the minimum is deleted
- Hollow nodes can be garbage collected and thrown if memory is needed
Hollow heaps in three steps

- Multiple root nodes
- One root node
- Two parents
- The purpose is to give you key insights what hollow heaps are about but not detailed proofs or implementation
- The exam may have a simple question about hollow heaps
Step 1: Multiple root nodes

- When an element is inserted, a new node is created
- This node becomes a new root
- It is then checked if this is the new minimum node
- We have a list of root nodes
Link operation

- Compare the keys of two nodes and make the one with smaller key the parent of the other
- The heap order of a tree is maintained using link operations
- A node has a single linked list of children
- A new child is inserted first in this list
Decrease-key operation

- If the element is a root, then the key is simply reduced — and check if this is the new min
- If not, a new root is created with the element
- The element is then moved from the previous node which becomes hollow
- Some of the children are moved to the new node as well
Delete operation

- If the deleted element is not the minimum, the node with it simply becomes hollow and we are done.
- If it is the minimum element, all hollow root nodes are destroyed by making their children new full root nodes.
- To reduce the number of root nodes, a number of link operations are performed.
Each node has a rank, which is a non-negative integer initially zero.
When reducing the number of hollow roots, link operations are performed on root nodes with the same rank.
The node which becomes the parent at a link has its rank incremented by one.
A node with rank $r$ has exactly $r$ children, except if $r > 2$ and the node has become hollow when the key of its former element was decreased.

In that case, the node has two children with ranks $r - 2$ and $r - 1$.

Let $r_u$ be the rank of $u$.

When an element is moved from a node $u$ to a node $v$ the rank of $v$ is set to $\max\{0, r_u - 2\}$.

All children of $u$ with rank less than $r_v$ are moved to $v$, with their children.

If the rank of $u$ is at least 2, then $u$ keeps two children with ranks $r - 2$ and $r - 1$.

If the rank of $u$ is one, then $u$ keeps its child (with rank zero).
Fibonacci numbers

- Recall Fibonacci numbers: $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$
- $F_0 = 0, F_1 = 1$ and $F_i = F_{i-1} + F_{i-2}$
- $F_{i+2} \geq \phi^i$ with $\phi = (1 + \sqrt{5})/2$
A node with rank $r$ has at least $F_{r+3} - 1$ descendants (both full and hollow).

We will show this using induction.

For $r = 0$ or $r = 1$ it is true.

For $r \geq 2$ the node itself and its children with ranks $r - 2$ and $r - 1$ are among the descendants.

From the induction hypothesis, the node has at least $1 + (F_{r+2} - 1) + (F_{r+1} - 1) = F_{r+3} - 1$ descendants.

Since $F_{r+3} - 1 \geq F_{r+2} \geq \phi^r$, the rank of a node in a heap with $N$ nodes (full or empty) is at most $\log_\phi N$. 

 Jonas Skeppstedt (jonasskeppstedt.net)
Efficient moving of children and efficient links

- The children of a node are stored in the order of decreasing rank.
- To move all except the first two children is therefore a constant time operation.
- When the minimum element is removed we need to find roots with the same rank in constant time.
- This is done using an array and the rank of a node as the index to the array.
- The first time you see a node with rank $r$ it is stored in the array at index $r$.
- The next time you see a node with rank $r$ you can therefore find it in constant time.
- Then you link and put back the new parent at index $r + 1$ and do a new link if any node already was stored at $r + 1$. 
Recall: deleting a non-minimum element is a constant time operation.

Deleting the minimum element is done by destroying hollow roots and then doing links to reduce the number of roots to at most \( \log N \).

To delete a hollow root and making its children new roots is a constant time operation.

The following can be shown:

- The worst case time of all hollow heap operations except delete take constant time.
- The amortized time of delete (and delete-min) takes \( O(\log N) \) on a heap with \( N \) nodes.

Thus: hollow heaps have constant time insert and reduce-key.

And array-based heaps instead have \( O(\log N) \) insert and reduce-key.

If insert and reduce-key are frequent, hollow heaps can be faster.
Step 2: One-root hollow heaps

- Allow links of nodes with different ranks
- By allowing this, it is possible to have only one root
- Now a child must be marked as coming either from a ranked or unranked link
- Either the heap is empty or the root is full (i.e. never a hollow root)
- When moving children of $u$ to $v$, all the unranked children of $u$ are always moved to $v$ plus the ranked children as before (i.e. keep one or two children in $u$)
Instead of moving some children of \( u \) to \( v \), \( v \) becomes a parent of \( u \)
That is, \( v \) becomes a second parent of \( u \)
Thus, the data structure is no longer a tree
It becomes a directed acyclic graph, or a dag
The heap order terminology is translated to dags
A child must have a key which is at least as big as the key of any of its parents
Observations

- A node in a two-parent hollow heap has at most one parent if it is full, and at most two parents if it is hollow.
- Motivation: there are only two ways to get a parent:
  1. a full root can get a first parent by becoming a child at a link, and
  2. a full node can become hollow at a decrease-key and get a second parent
  3. a hollow node cannot become full and therefore not get any additional parent
Implementations

- Array-based
- Fibonacci heap
- Two-root hollow heap
- Note insert and decrease_key (i.e. change_position in array-based)